A Survey on the Performance of Parallel Downloading

J. L. Chiang

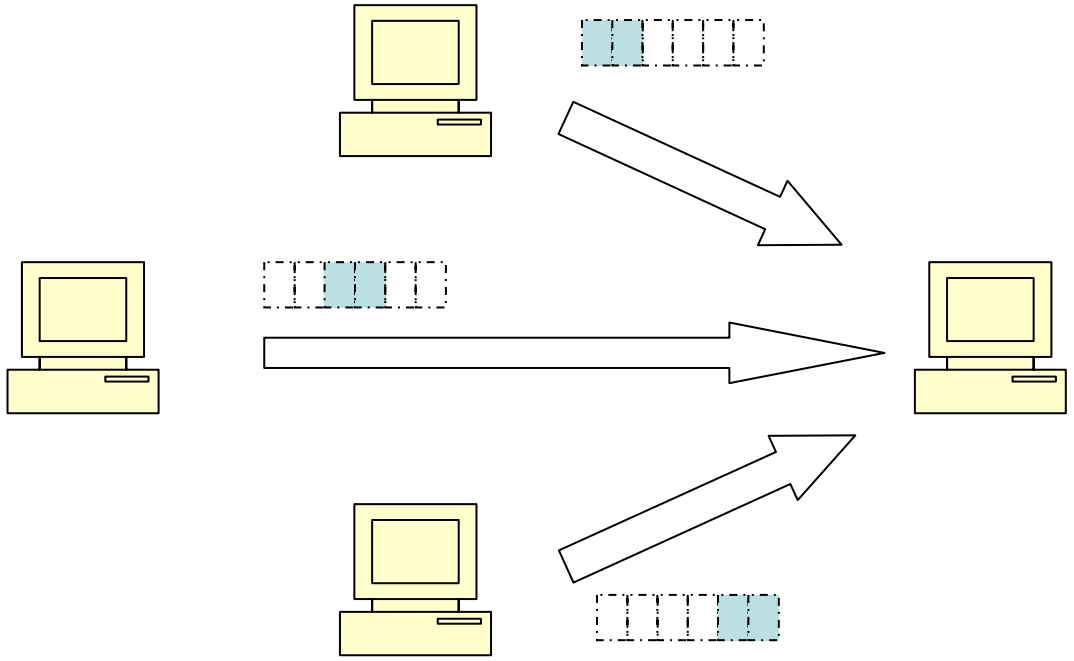
April 7, 2005

Outline

- Parallel download schemes
  - Static equal
  - Static unequal
  - Dynamic
- Performance comparison and issues
- adPD scheme
- Large-scale deployment of PD
- Conclusions

Static Equal

- The Client downloads an equal portion of the desired file from each of the available servers.



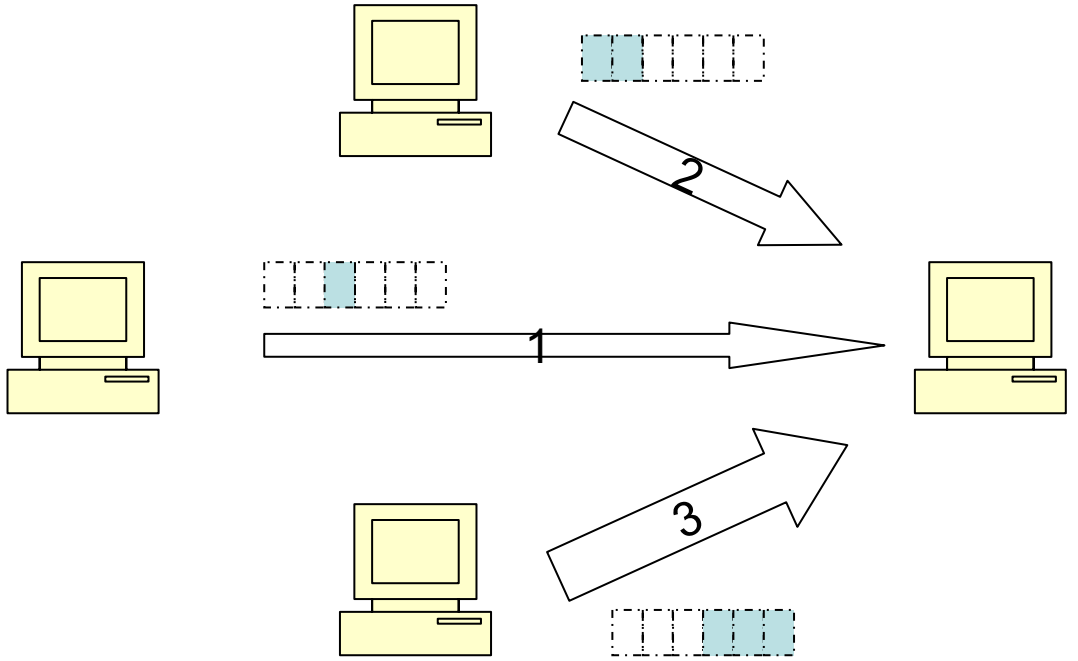
Static Equal

- *L*: size of file;
- *n*: # of servers;
- *bi*: transmission rate of server *i;*
- b_{bad}: transmission rate of the slowest server;
- $t_{opt} = L/\sum bi$;
- $t_{bad} = (L/n)b_{bad}$;
- Performance ratio:

$$k = \frac{t_{bad}}{t_{opt}} = \frac{\sum_i b_i}{n b_{bad}}$$

Static Unequal

- The client request different amount of data proportional to each server's (estimated) transmission rate.



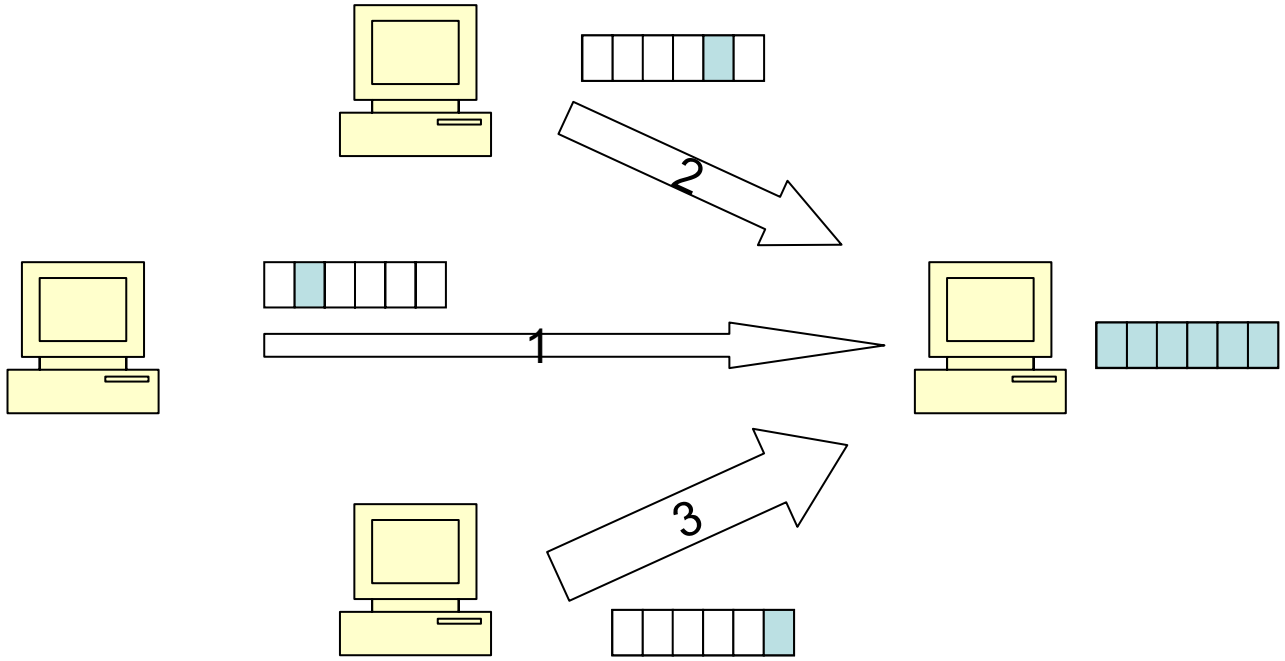
Static Unequal

- *Li* = *L*(*bi*/ $\sum$ *bi*): the amount of data assigned for server *i*;
- *b'i* : estimate of transmission rate of server *i*;
- Performance ratio:

$$k = \frac{\sum_i b_i}{\sum_i b'_i} \cdot \max_i \frac{b'_i}{b_i}$$

Dynamic

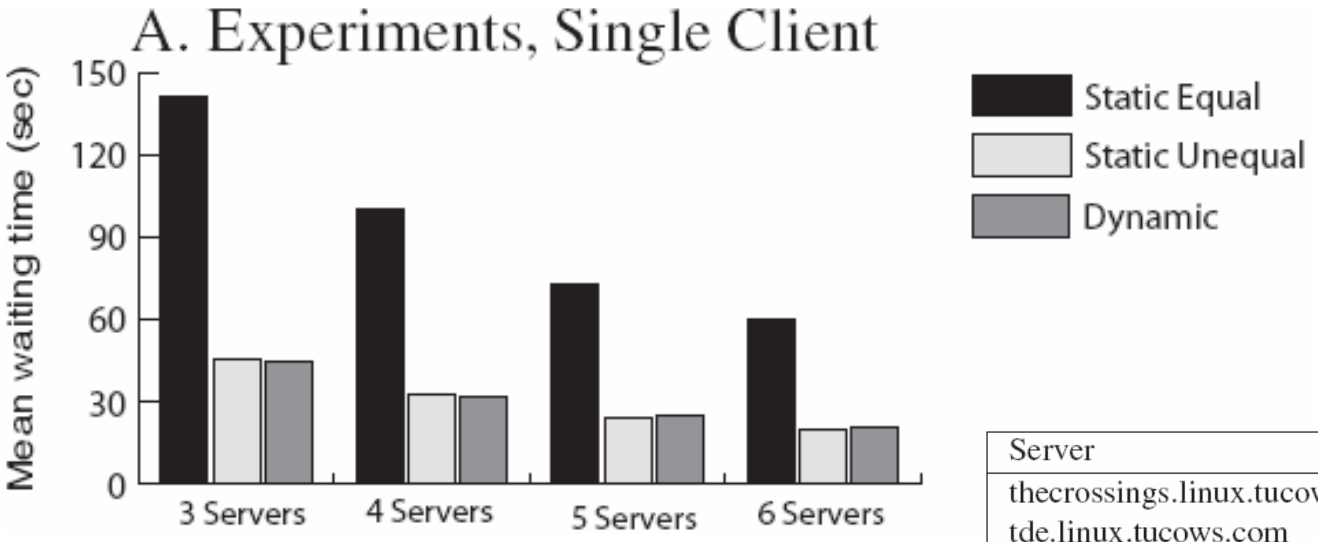
- The file is divided into equal blocks and the client requests a block from each of the servers. When a server finishes, then the client asks for another block until receiving all pieces.



Dynamic

- ℓ : size of block;
- $B = L/\ell$: # of blocks;
- *Bi* is proportional to the transmission rate of server *i* when $\ell << L$, and *k* -> 1 as static unequal.
- Termination idle time <= ℓ / b_{bad};

Internet Experiments [1]



Server	Location
thecrossings.linux.tucows.com	Mountain View, CA
tde.linux.tucows.com	Wheat Ridge, CO
delaware.linux.tucows.com	Dover, DE
intermarine.linux.tucows.com	Jacksonville, FL
agentware.linux.tucows.com	Atlanta, GA
nitco.linuxberg.com	Indiana
linuxberg.mv.net	Londonderry, NH
epix.linux.tucows.com	Dallas, PA
config.linux.tucows.com	Ravenna, OH
emperor.linux.tucows.com	Austin, TX
datasync.linux.tucows.com	Biloxi, MS
infostreet.linux.tucows.com	Tarzana, VA

Static Equal

- Schedule at startup
- Adv.
  - Simple
- Disadv.
  - Performance achieved is far from optimal
  - Increasing servers also increases the gap between optimal time and achieved one
- k->1.01~11.84(2 servers)
- k->1.45~20.68(9 servers)
- 60-70% > twice

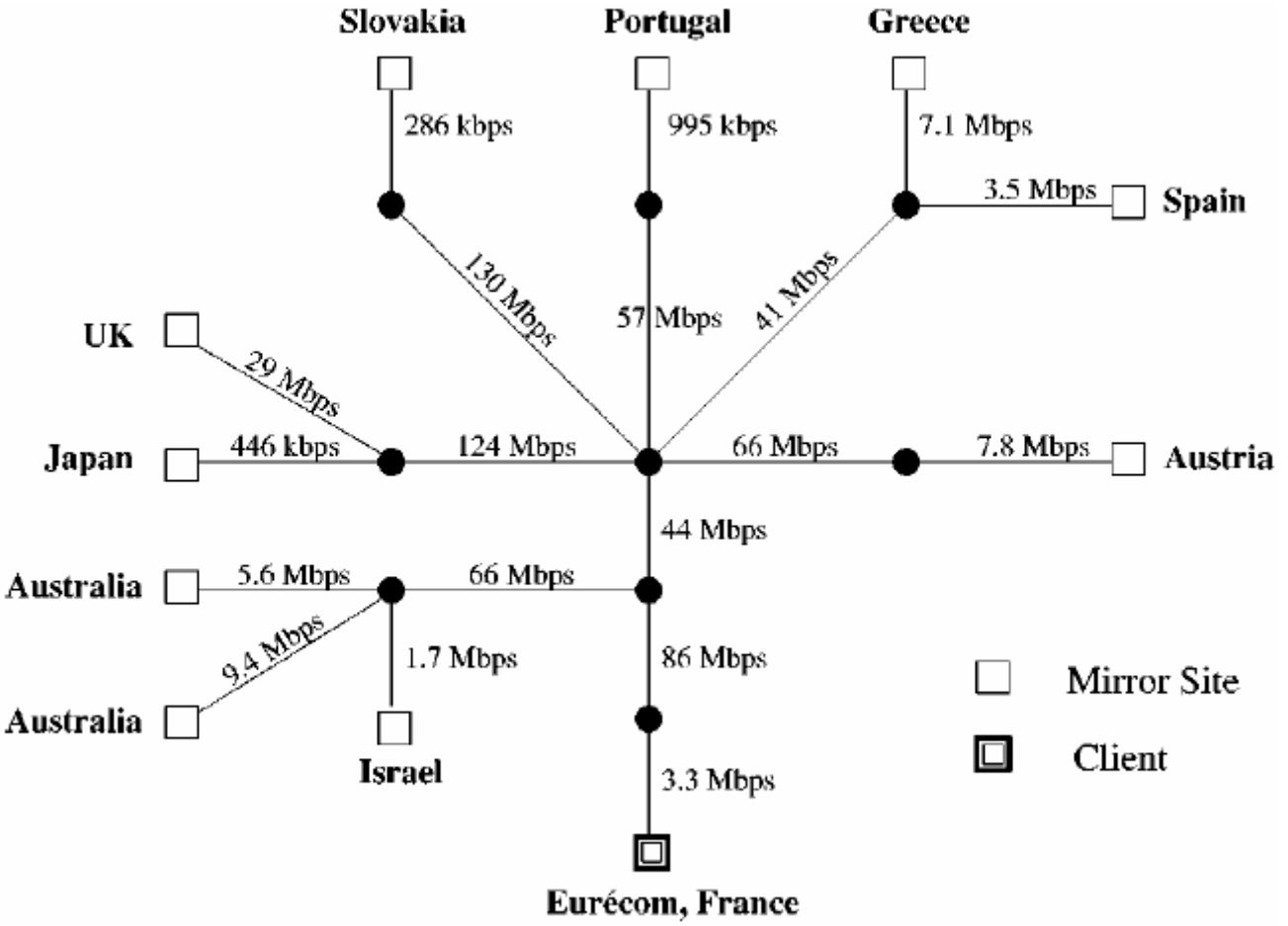
Static Unequal

- Schedule at startup
- Adv.
  - Assign more data to faster servers to decrease overall download time
- Disadv.
  - Require a database of server rates
  - difficult to estimate transmission rate

$$b_i^n = \alpha b_i^{n-1} + (1-\alpha)\, b_i^{real}$$

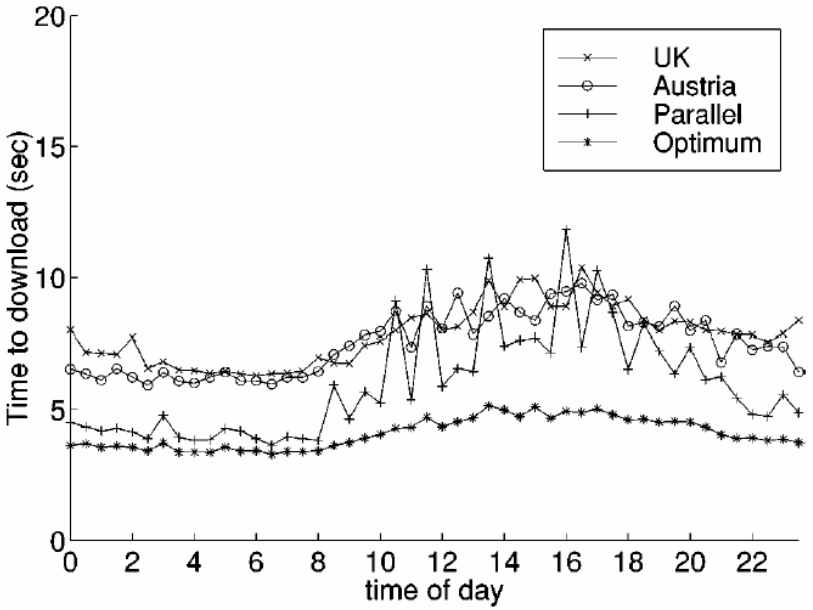
- K->1.00~1.93(2 servers)
- K->1.09~5.75(9 servers)
- 10-30% > twice

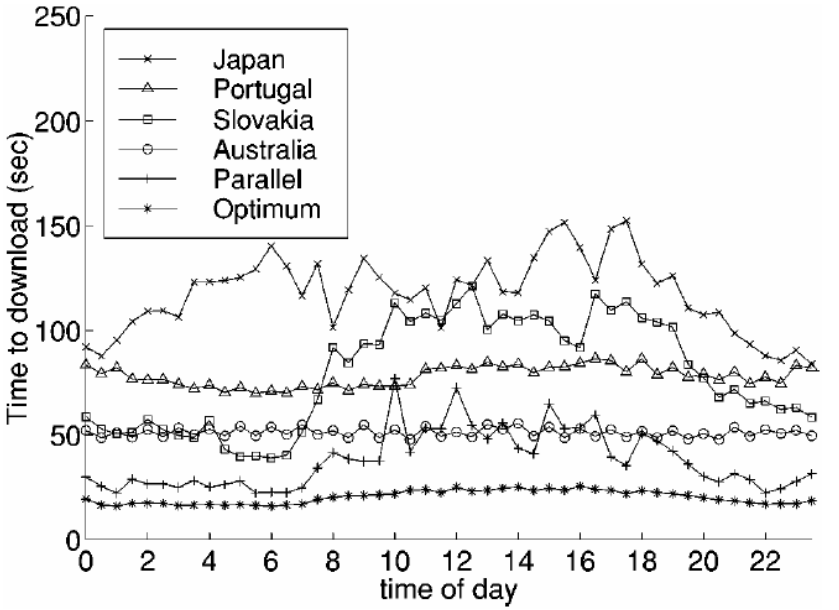
Another Experiment Setup [2]



Static Unequal

- Rapid changes in busy hours result in poor estimates of servers' rates. (763kB)

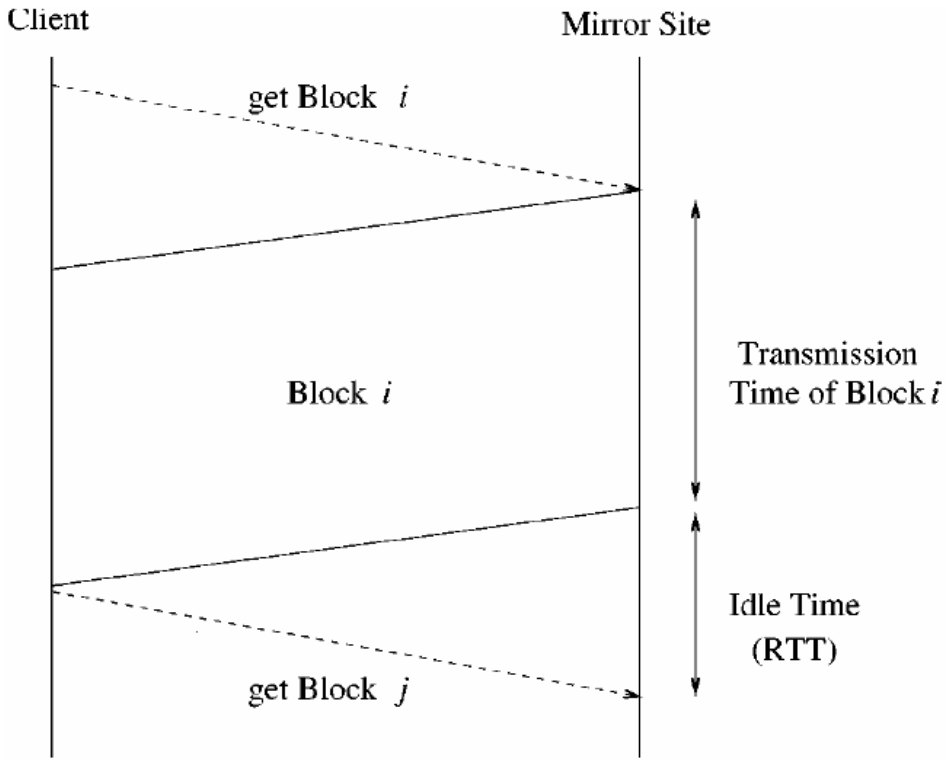




Dynamic

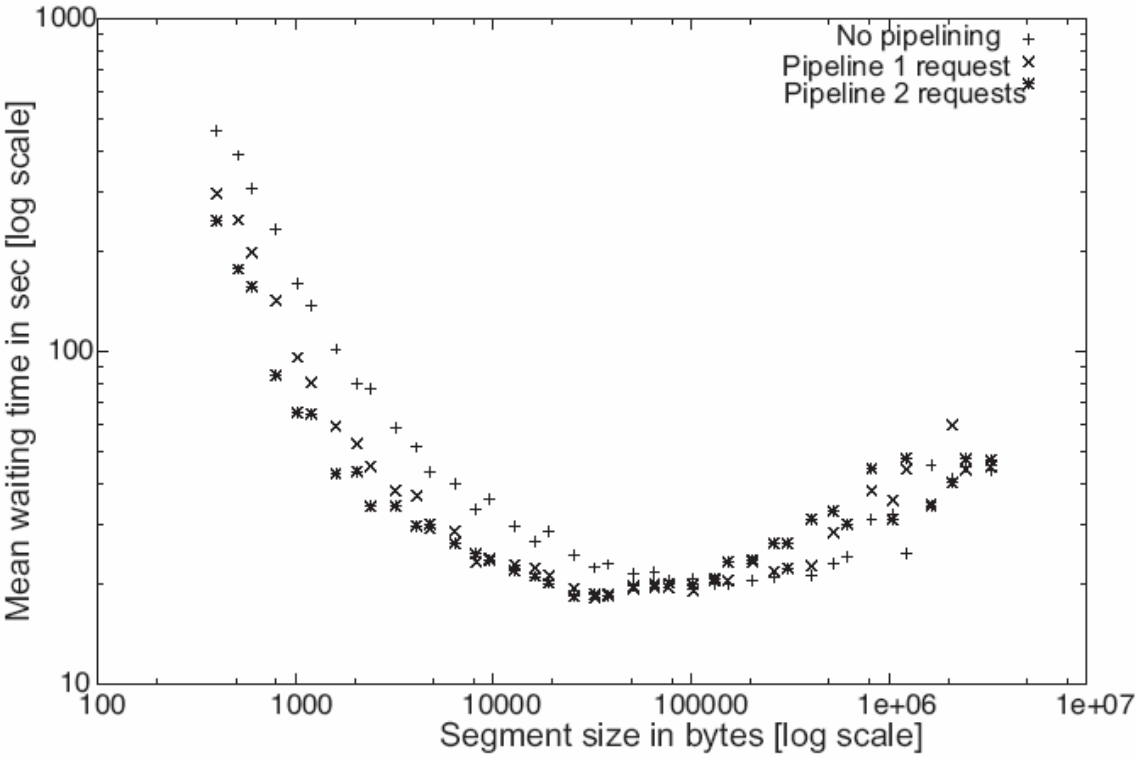
- Dynamic schedule
- Adv.
  - Assign more data from servers that are currently performing better to decrease overall download time
- Disadv.
  - Addition overhead during transmission
  - Interblock idle time
  - Termination idle time

Interblock Idle Time



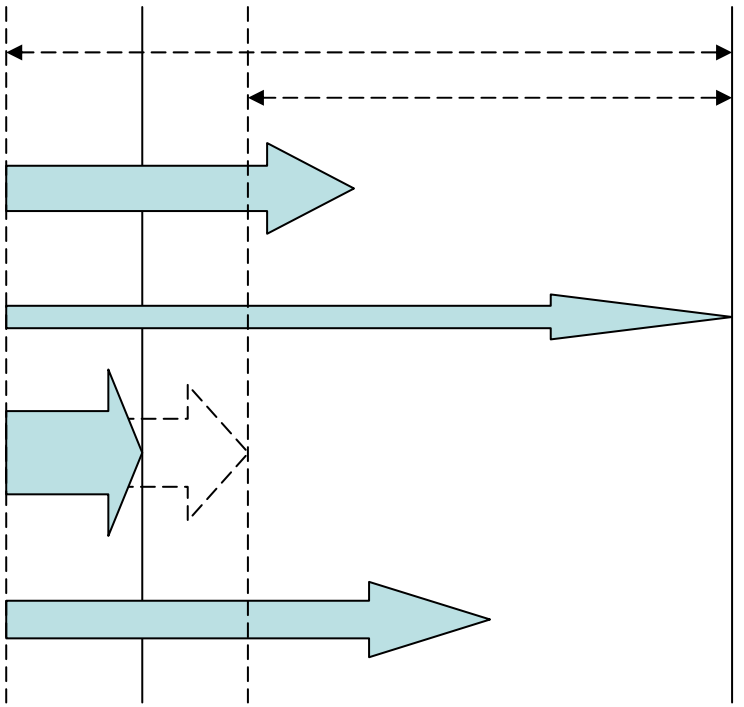
Request Pipelining

- 7.1MB, 8 servers



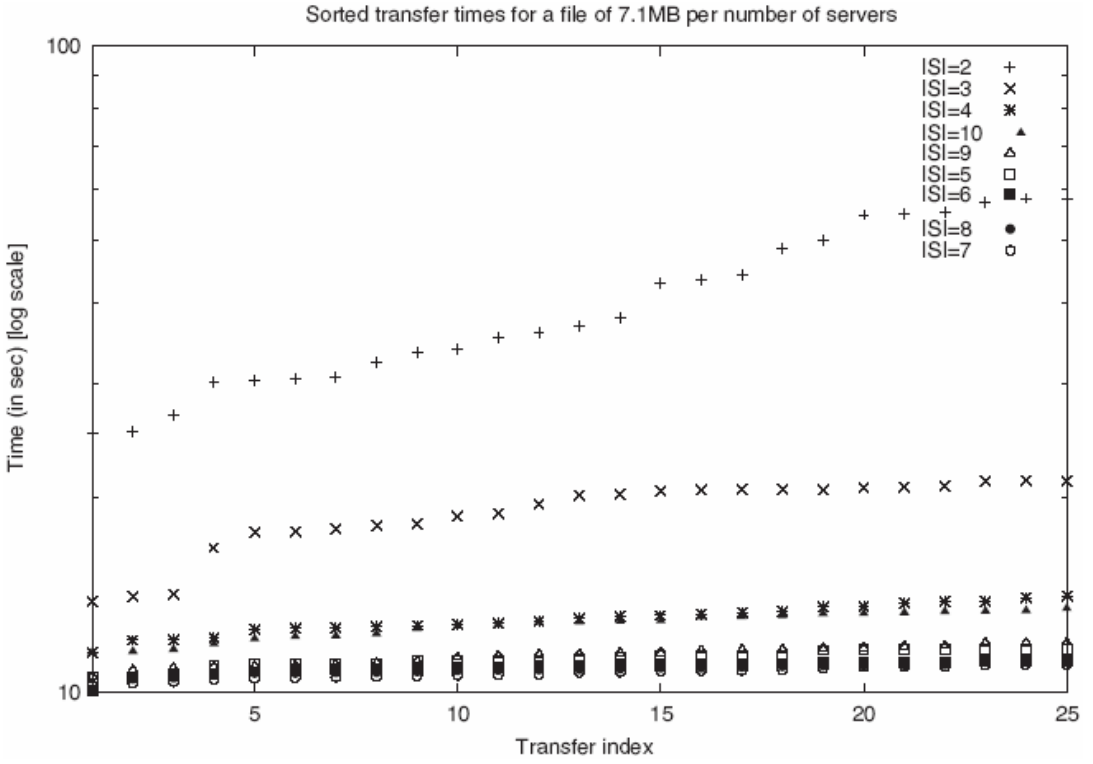
Termination Idle Time

- Termination idle time <= ℓ / b_{bad}



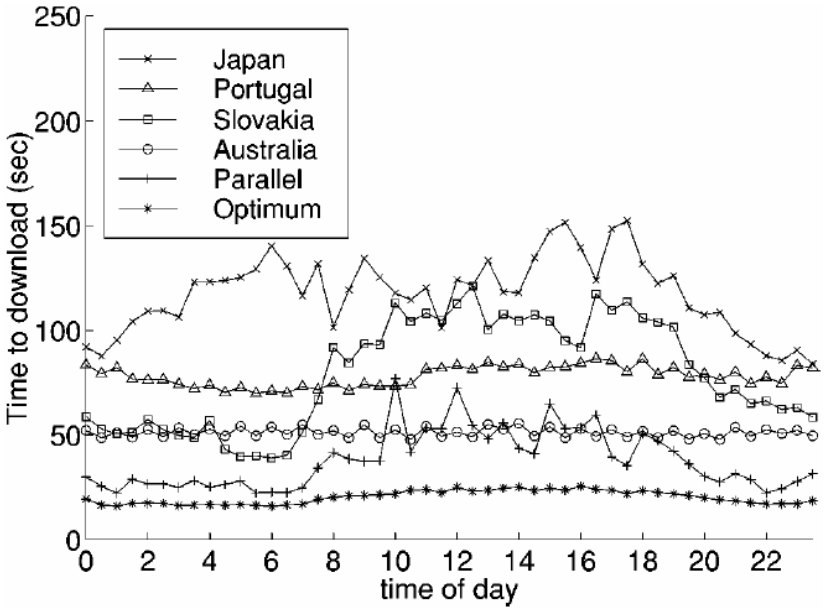
Top 25

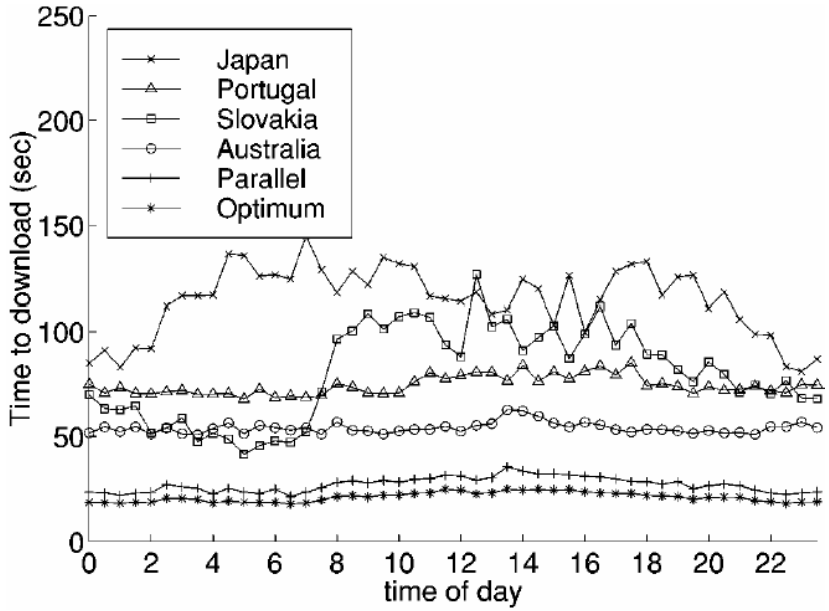
- 7.1MB, 2-10 servers



Static Unequal v.s. Dynamic

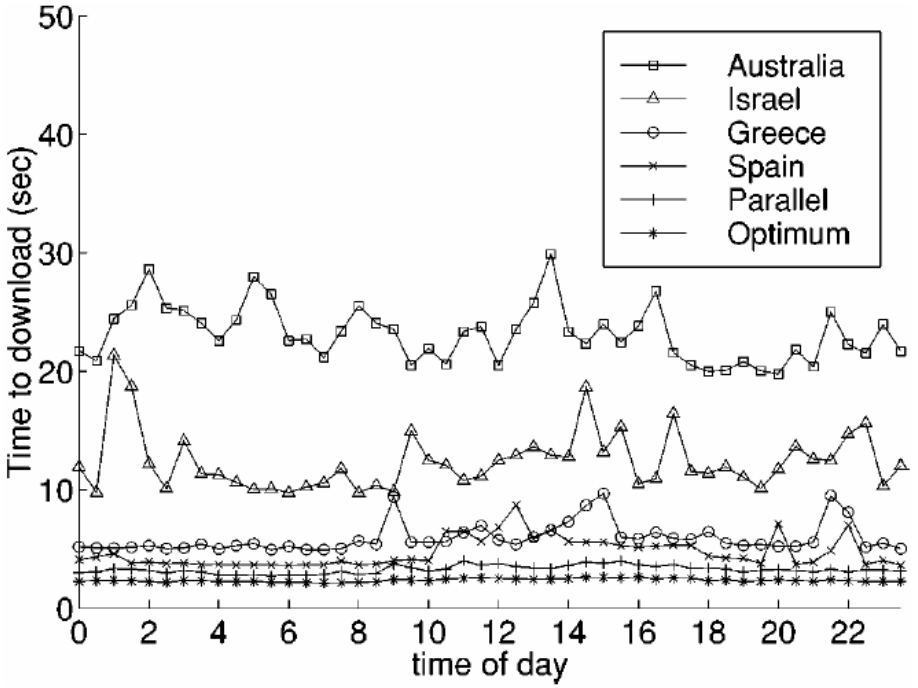
- 763kB, L/ℓ =30, 4 servers





Server Heterogeneity

- 256kB, L/ℓ =30, 2 fast and 2 slow servers



Issues

- How to divide the file into blocks
- How to allocate the downloading jobs among all connections
- ℓ : size of block
  - A fine granularity is achieved with small values of ℓ;
  - Small ℓ will cause more block request sending to server and burden more overhead on the server and network;

adPD Algorithm [3]

Suppose $node_{PD}$ opens m connections to m replicas of file f whose size is F, and the speed of each connection is $v_i(t), i = 1,\ldots,m$. The process of adPD scheme is as follow:

1. $node_{PD}$ divides f into m equal parts, and retrieves data respectively from m connections.
2. At time spot t, when a connection i finishes its current job, $node_{PD}$ reallocates part of unfinished work of one certain connection to it, the reassignment is proportional to the speed of two connections:
   a) $node_{PD}$ calculates the weighted average speed v_j of each connection first:

$$v_j = \alpha F_j / t + (1-\alpha) v_j^{now},\ 0 < \alpha < 1,\ j = 1,\ldots,m$$

   F_j is the total amount of finished bytes by connection j, $v_j^{now} = v_j(t)$ is connection j's current speed.
   b) By using v_j, $node_{PD}$ can estimates the remaining time for each connection to complete their unfinished job, and chooses a connection j which has longest remaining time and whose weighted average speed is slower than connection i.

adPD Algorithm [3]

Then $node_{PD}$ reallocates part of the unfinished job $\overline{F}_j$ of connection j to connection i according to next formula:

$$\overline{F}_i / \overline{F}'_j = v_i / v_j \,, \quad \overline{F}_j = \overline{F}'_j + \overline{F}_i \tag{3}$$

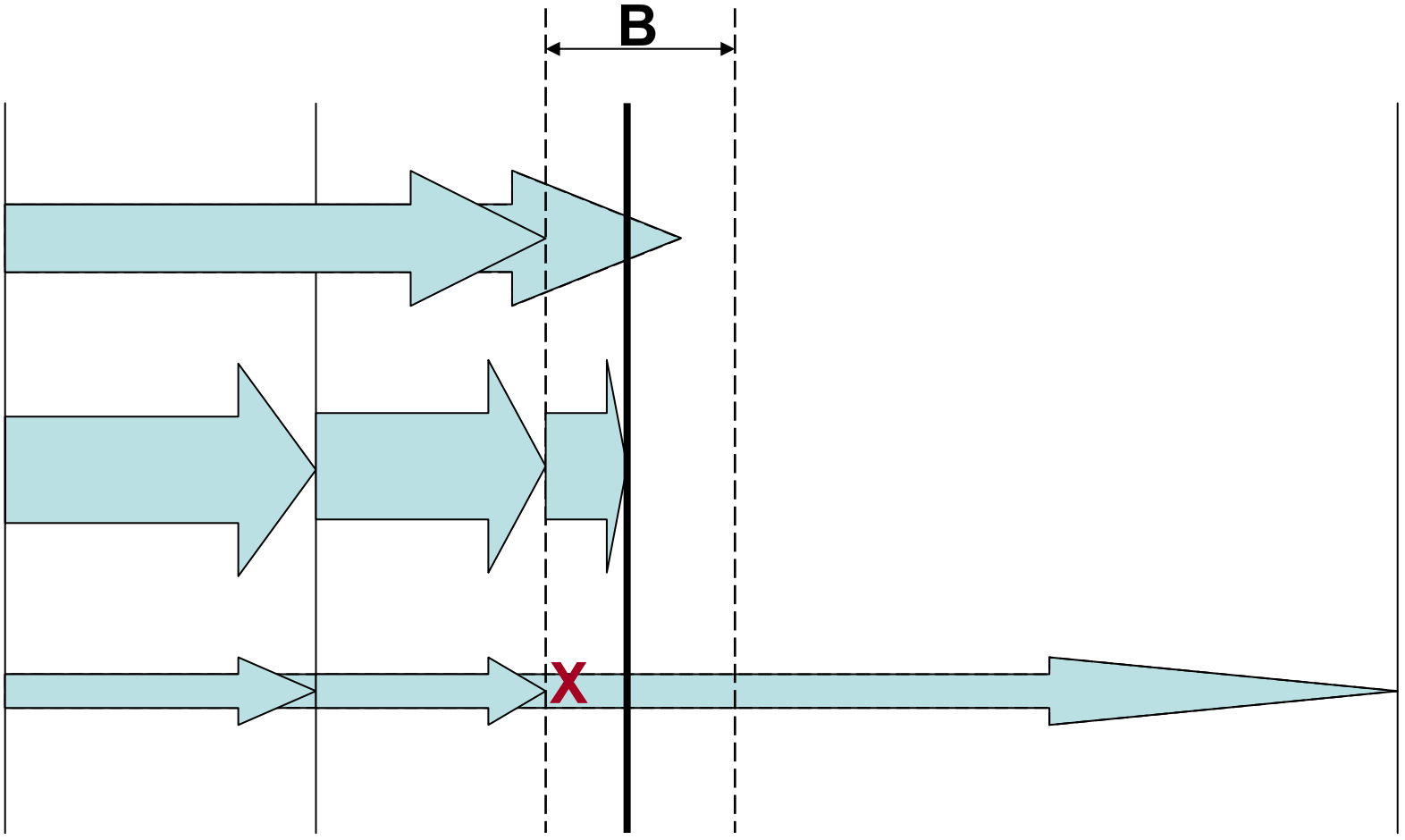
Where $\overline{F}'_j$ is the unfinished job of connection j after reallocation.

If all slower connections have done their work, connection i chooses the slowest one of the connections whose speed are faster than it, and tries to help that connection with its unfinished job using formula (3).

c) The reallocation can not do without limitation. On the condition of connection j's unfinished job is fewer than B, where $B \ll F$, then if speed of j is faster than i, j will reject the help of i; or if j is slower than i, j will give up its unfinished job and let i do its job.

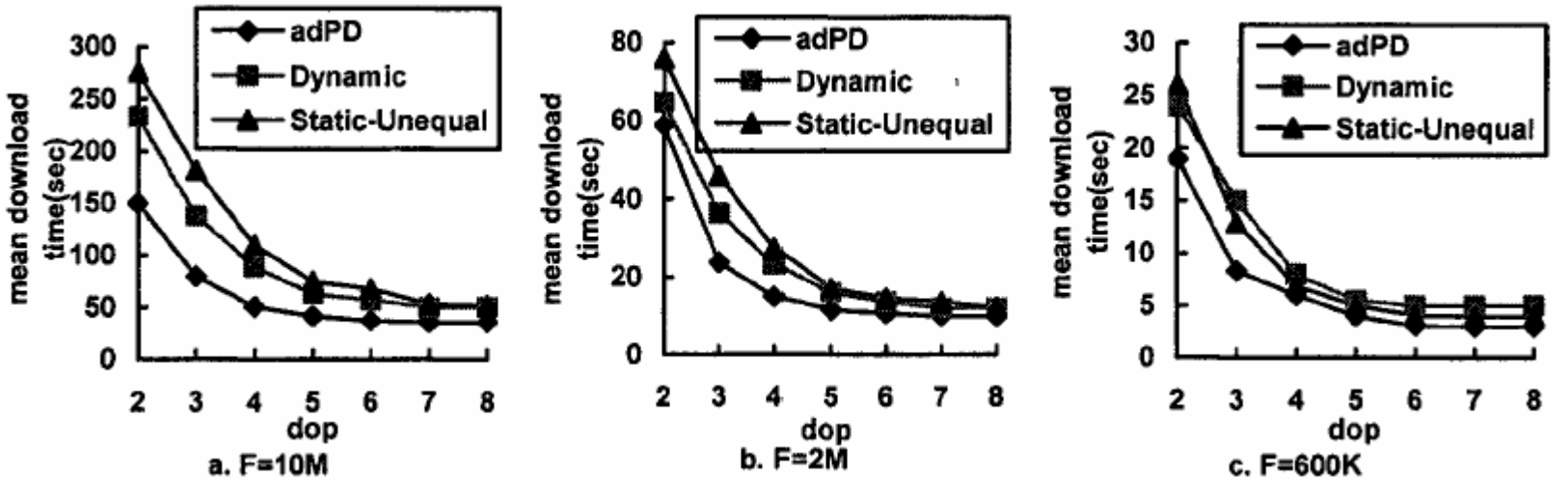
3. When $node_{PD}$ gets all the pieces, it emerges them and get the whole file.

adPD Illustration



adPD Performance

- 50 nodes, **B**=50K, α =0.59



adPD Advantages

- It can adjust the size of transferring data block dynamically.
- It ensures the continuity of downloading processes of every connection.
- It adapts to variable speed of connections.
- It minimize the termination idle time by **B**.

Large-scale Deployment of PD

- Single client v.s. multiple clients
- Server load: *m*/*n* -> *m*

- PD can actually lead to a degradation on performance when it is performed by all or a large proportion of clients.
- PD clients experience improvement at the expense of clients without PD capability (unfairness).

Simulation Setup

- ns2 simulation
- GT-ITM random topology of 100 routers
- 10 servers
- 100, 200, 300 clients
- 1MB file
- Static equal, static unequal, dynamic, simple

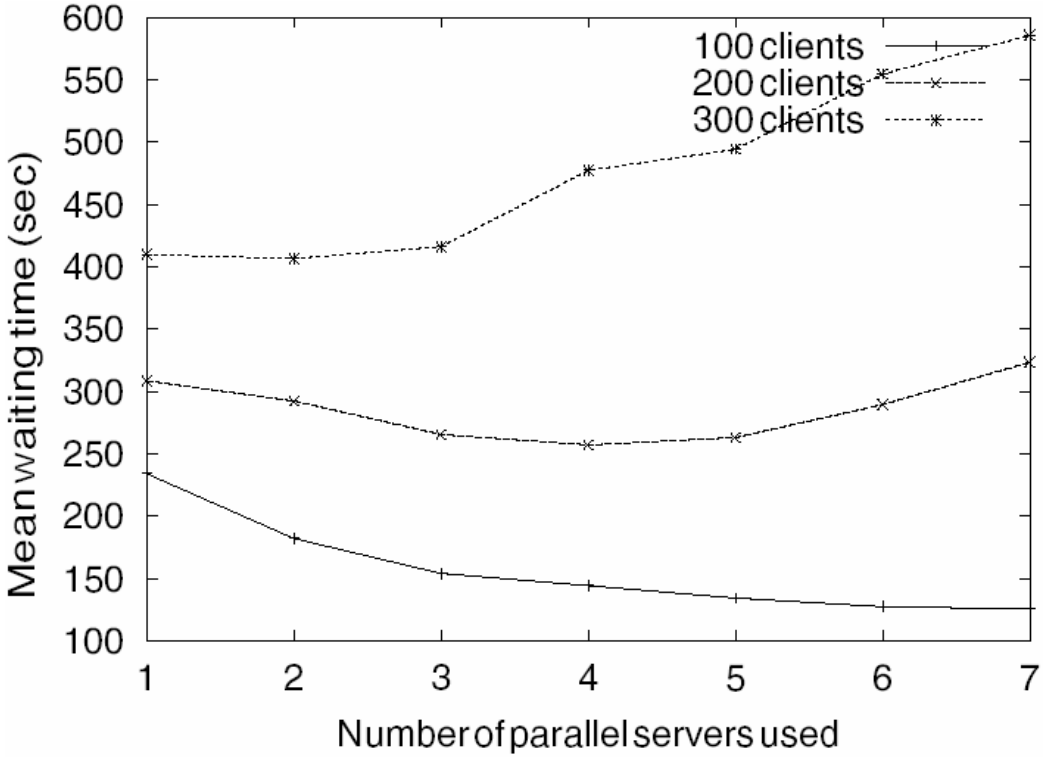
Simulation with Multiple Clients

100 clients

Sources of Degradation

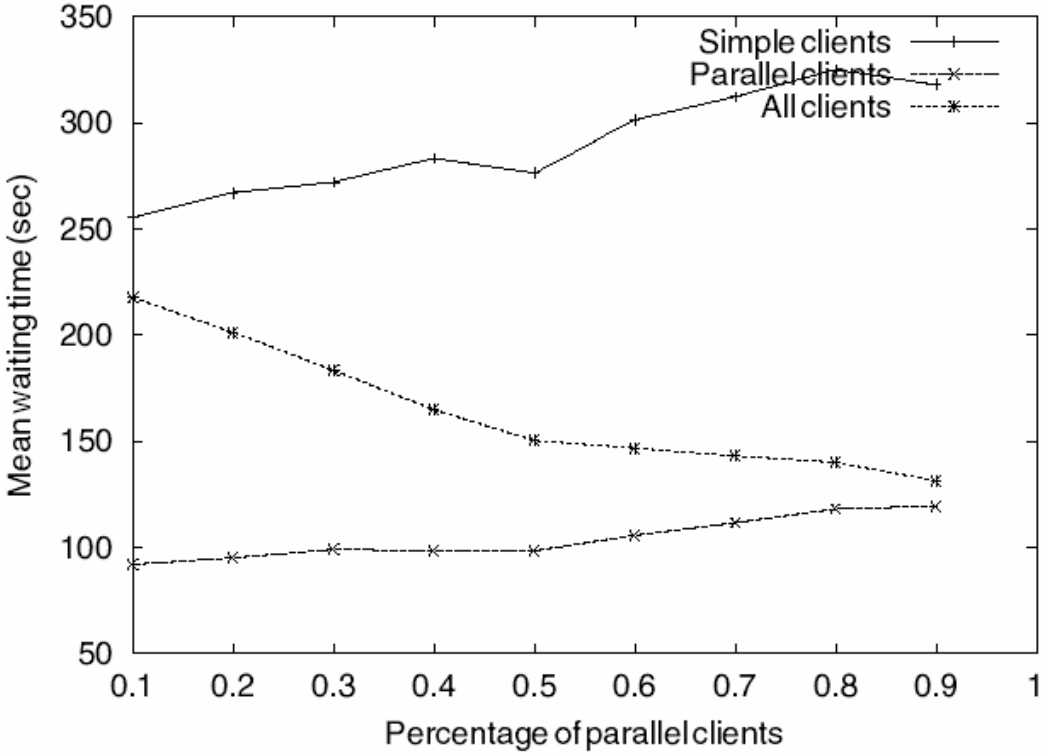
- Clients and servers require extra overheads in opening and maintaining TCP connections.
- More TCP flows are competing in the network.
- More requests and data packets are injected into the network.

Degradation with More Clients



Unfairness to Simple Clients

- 100 clients with 10 servers



Conclusions

- The performance advantage of using parallel downloading techniques as seen by a client will drop as more users employ them.
- The improvement of parallel downloading comes at the expense of clients that do not have such a capability.
- The evaluation of new schemes to be used in the Internet should be carefully considered in large-scale deployment instead of in the view of a single client.

Conclusions

- Parallel downloading outperforms single downloading in peer-to-peer scenarios:
  - Peers are not dedicated to acting as file servers.
  - Peer heterogeneity may produce large fluctuation on performance.
  - Network conditions of peers are much more unpredictable than dedicated servers.
  - Dynamic joining and leaving of peers impair availability of resources.
- Parallel downloading in P2P requires modifications of those solutions developed for mirrored servers..

References

- C. Gkantsidis, M. Ammar, and E. Zegura, “On the Effect of Large-scale Deployment of Parallel Downloading,” *IEEE WIAPP’03*.
- P. Rodriguez and E. W. Biersack, “Dynamic Parallel Access to Replicated Content in the Internet,” *IEEE/ACM Transaction on Networking*, Aug. 2002.
- X. Zhou, X. L. Lu, M. S. Hou, and C. Zhan, “a Speed-based Adaptive Dynamic Parallel Downloading Technique,” *ACM SIGOPS Operating Systems Review*, Jan. 2005.